

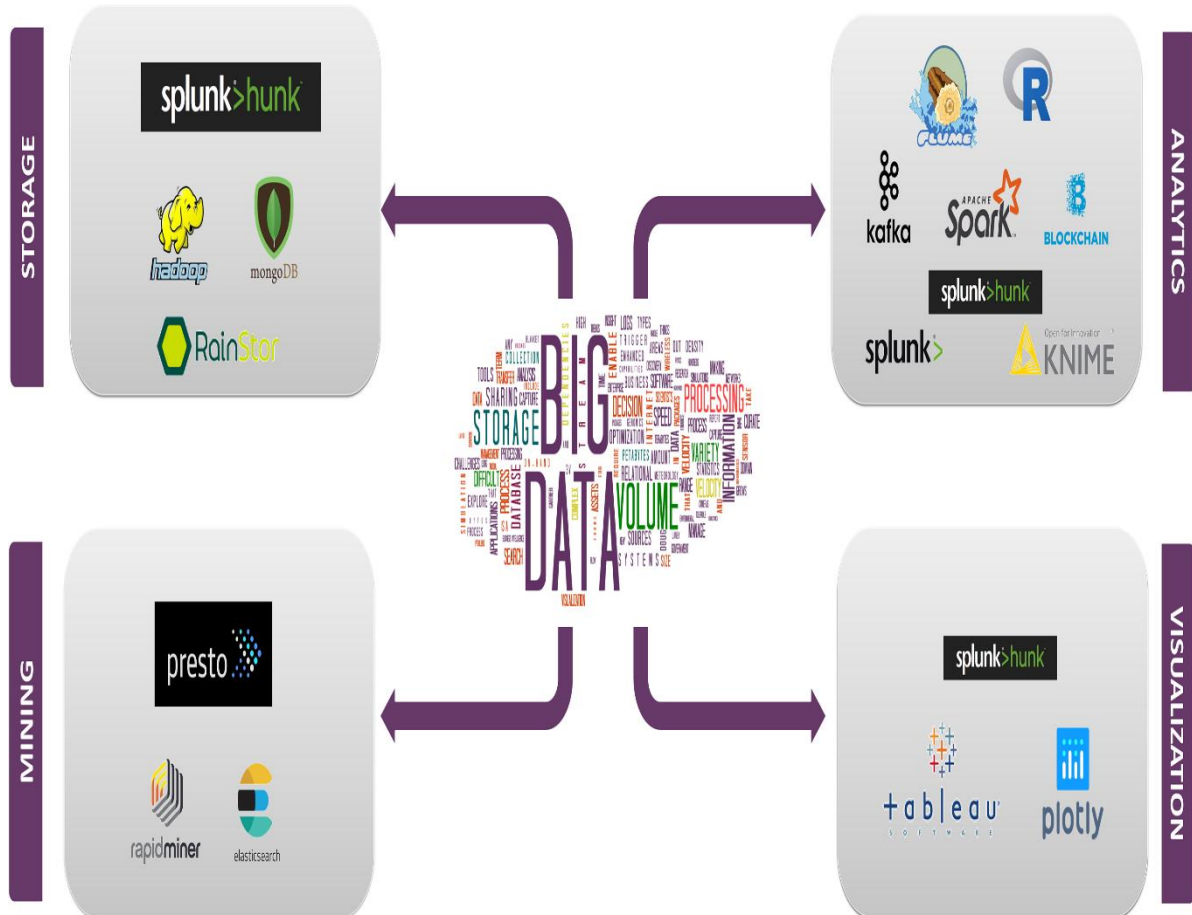
1) Describe various Big Data Technologies and Tools.

Ans:

Top Big Data Technologies

Top big data technologies are divided into **4** fields which are classified as follows:

- **Data Storage**
- **Data Mining**
- **Data Analytics**
- **Data Visualization**



Now let us deal with the technologies falling under each of these categories with their facts and capabilities, along with the companies which are using them.

Let us get started with **Big Data Technologies in Data Storage.**

Data Storage

1.Hadoop



Hadoop Framework was designed to store and process data in a **Distributed Data Processing Environment** with commodity hardware with a simple programming model. It can Store and Analyse the data present in different machines with High Speeds and Low Costs.
Developed by: Apache Software Foundation in the year 2011 10th of Dec.

Written in: JAVA

Current stable version: Hadoop 3.11

2.MongoDB



The **NoSQL** Document Databases like **MongoDB**, offer a direct alternative to the rigid schema used in Relational Databases. This allows **MongoDB** to offer Flexibility while handling a wide variety of **Datatypes** at large volumes and across **Distributed Architectures**.

Developed by: MongoDB in the year 2009 11th of Feb

Written in: C++, Go, JavaScript, Python

Current stable version: MongoDB 4.0.10

3.Rainstor



RainStor is a software company that developed a Database Management System of the same name designed to Manage and Analyse Big Data for large enterprises. It uses **Deduplication Techniques** to organize the process of storing large amounts of data for reference.

Developed by: RainStor Software company in the year 2004.

Works like: SQL

Current stable version: RainStor 5.5

4.Hunk



Hunk lets you access data in remote Hadoop Clusters through virtual indexes and lets you use the Splunk Search Processing Language to analyse your data. With Hunk, you can Report and Visualize large amounts from your Hadoop and NoSQL data sources.

Developed by: Splunk INC in the year 2013.

Written in: JAVA

Current stable version: Splunk Hunk 6.2

Big Data Technologies used in Data Mining.

Data Mining

1.Presto



Presto is an open source **Distributed SQL Query Engine** for running **Interactive Analytic Queries** against data sources of all sizes ranging from Gigabytes to Petabytes. Presto allows querying data in **Hive**, **Cassandra**, **Relational Databases** and **Proprietary Data Stores**.

Developed by: Apache Foundation in the year 2013.

Written in: JAVA

Current stable version: Presto 0.22

2.Rapid Miner



RapidMiner is a Centralized solution that features a very powerful and robust Graphical User Interface that enables users to Create, Deliver, and maintain Predictive Analytics. It allows creating very Advanced Workflows, Scripting support in several languages.

Developed by: RapidMiner in the year 2001

Written in: JAVA

Current stable version: RapidMiner 9.2

3.Elasticsearch



Elasticsearch is a Search Engine based on the Lucene Library. It provides a Distributed, MultiTenant-capable, Full-Text Search Engine with an HTTP Web Interface and Schema-free JSON documents.

Developed by: Elastic NV in the year 2012.

Written in: JAVA

Current stable version: ElasticSearch 7.1

Big Data Technologies used in Data Analytics.

Data Analytics

1.Kafka



Apache Kafka is a Distributed Streaming platform. A streaming platform has Three Key Capabilities that are as follows:

- Publisher
- Subscriber
- Consumer

This is similar to a Message Queue or an Enterprise Messaging System.

- **Developed by:** Apache Software Foundation in the year 2011
- **Written in:** Scala, JAVA
- **Current stable version:** Apache Kafka 2.2.0

2.Splunk



Splunk captures, Indexes, and correlates Real-time data in a Searchable Repository from which it can generate Graphs, Reports, Alerts, Dashboards, and Data Visualizations. It is also used for Application Management, Security and Compliance, as well as Business and Web Analytics.

Developed by: Splunk INC in the year 2014 6th May

Written in: AJAX, C++, Python, XML

Current stable version: Splunk 7.3

3.KNIME



KNIME allows users to visually create Data Flows, Selectively execute some or All Analysis steps, and Inspect the Results, Models, and Interactive views. **KNIME** is written in Java and based on Eclipse and makes use of its Extension mechanism to add Plugins providing Additional Functionality.

Developed by: KNIME in the year 2008

Written in: JAVA

Current stable version: KNIME 3.7.2

4. Spark



Spark provides **In-Memory Computing** capabilities to deliver Speed, a Generalized Execution Model to support a wide variety of applications, and **Java**, **Scala**, and **Python** APIs for ease of development.

Developed by: Apache Software Foundation

Written in: Java, Scala, Python, R

Current stable version: Apache Spark 2.4.3

5. R-Language



R is a Programming Language and free software environment for **Statistical Computing** and **Graphics**. The **R** language is widely used among Statisticians and Data Miners for developing Statistical Software and majorly in Data Analysis.

Developed by: R-Foundation in the year 2000 29th Feb

Written in: Fortran

Current stable version: R-3.6.0

6. Blockchain



BlockChain is used in essential functions such as payment, escrow, and title can also reduce fraud, increase financial privacy, speed up transactions, and internationalize markets.

BlockChain can be used for achieving the following in a Business Network Environment:

Shared Ledger: Here we can append the Distributed System of records across a Business network.

Smart Contract: Business terms are embedded in the transaction Database and Executed with transactions.

Privacy: Ensuring appropriate Visibility, Transactions are Secure, Authenticated and Verifiable

Consensus: All parties in a Business network agree to network verified transactions.

- **Developed by:** Bitcoin
- **Written in:** JavaScript, C++, Python
- **Current stable version:** Blockchain 4.0

With this, we shall move into **Data Visualization Big Data technologies**

Data Visualization

7. Tableau



Tableau is a Powerful and Fastest growing Data Visualization tool used in the **Business Intelligence** Industry. Data analysis is very fast with **Tableau** and the Visualizations created are in the form of Dashboards and Worksheets.

Developed by: Tableau 2013 May 17th

Written in: JAVA, C++, Python, C

Current stable version: Tableau 8.2

8. Plotly



Mainly **used** to make creating Graphs faster and more efficient. API libraries for **Python, R, MATLAB, Node.js, Julia**, and **Arduino** and a **REST API**. **Plotly** can also be used to style **Interactive Graphs** with **Jupyter notebook**.

Developed by: Plotly in the year 2012

Written in: JavaScript

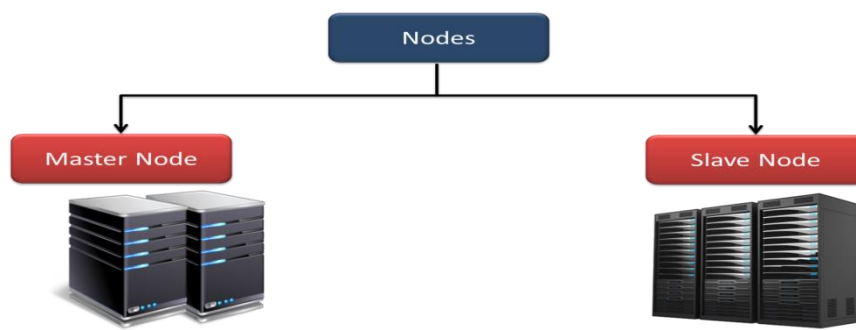
Current stable version: Plotly 1.47.4

2. Describe the roles of Name node and Data node.

Ans:

HDFS Nodes

Hadoop works in **master-slave** fashion, HDFS also has 2 types of nodes that work in the same manner. There are **namenode(s)** and **datanodes** in the cluster.



HDFS Master (Namenode)

Namenode regulates file access to the clients. It maintains and manages the slave nodes and assign tasks to them. Namenode executes file system namespace operations like opening, closing, and renaming files and directories. It should be deployed on reliable hardware.

It is also known as *Master* node. NameNode does not store actual data or dataset. NameNode stores Metadata i.e. number of **blocks**, their location, on which Rack, which Datanode the data is stored and other details. It consists of files and directories.

Tasks of HDFS NameNode

- Manage file system namespace.
- Regulates client's access to files.
- Executes file system execution such as naming, closing, opening files and directories.

HDFS Slave (Datanode)

There are n number of slaves (where n can be upto 1000) or data nodes in Hadoop Distributed File System which manage storage of data. These slave nodes are the actual worker nodes which do the tasks and serve read and write requests from the file system's clients. They also perform block creation, deletion, and replication upon instruction from the NameNode. Once a block is written on a datanode, it replicates it to other datanode and process continues until the number of replicas mentioned is created. Datanodes can be deployed on commodity Hardware and we need not deploy them on very reliable hardware.

It is also known as *Slave*. HDFS Datanode is responsible for storing actual data in HDFS. Datanode performs **read and write operation** as per the request of the clients. Replica block of Datanode consists of 2 files on the file system. The first file is for data and second file is for recording the block's metadata. HDFS Metadata includes checksums for data. At startup, each Datanode connects to its corresponding Namenode and does handshaking. Verification of namespace ID and software version of DataNode take place by handshaking. At the time of mismatch found, DataNode goes down automatically.

Tasks of HDFS DataNode

- DataNode performs operations like block replica creation, deletion, and replication according to the instruction of NameNode.
- DataNode manages data storage of the system.

This was all about HDFS as a Hadoop Ecosystem component.

Hadoop HDFS Daemons

There are 2 daemons which run for HDFS for data storage:

- **Namenode:** This is the daemon that runs on all the masters. Name node stores metadata like filename, the number of blocks, number of replicas, a location of blocks, block IDs etc. This metadata is available in memory in the master for faster retrieval of data. In the local disk, a copy of metadata is available for persistence. So name node memory should be high as per the requirement.
- **Datanode:** This is the daemon that runs on the slave. These are actual worker nodes that store the data.

3. Explain about parallel copying with distcp.

Ans:

Parallel Copying with distcp

The HDFS access patterns that we have seen so far focus on single-threaded access. It's possible to act on a collection of files — by specifying file globs, for example — but for efficient parallel processing of these files, you would have to write a program yourself. Hadoop comes with a useful program called distcp for copying data to and from Hadoop filesystems in parallel.

One use for distcp is as an efficient replacement for `hadoop fs -cp`. For example, you can copy one file to another with:[34]

```
% hadoop distcp file1 file2
```

You can also copy directories:

```
% hadoop distcp dir1 dir2
```

If dir2 does not exist, it will be created, and the contents of the dir1 directory will be copied there. You can specify multiple source paths, and all will be copied to the destination.

If dir2 already exists, then dir1 will be copied under it, creating the directory structure dir2/dir1.

If this isn't what you want, you can supply the `-overwrite` option to keep the same directory structure and force files to be overwritten. You can also update only the files that have changed using the `-update` option. This is best shown with an example. If we changed a file in the dir1 subtree, we could synchronize the change with dir2 by running:

```
% hadoop distcp -update dir1 dir2
```

distcp is implemented as a MapReduce job where the work of copying is done by the maps that run in parallel across the cluster. There are no reducers. Each file is copied by a single map, and distcp tries to give each map approximately the same amount of data by bucketing files into roughly equal allocations. By default, up to 20 maps are used, but this can be changed by specifying the `-m` argument to distcp. A very common use case for distcp is for transferring data between two HDFS clusters. For example, the following creates a backup of the first cluster's /foo directory on the second:

```
% hadoop distcp -update -delete -p hdfs://namenode1/foo hdfs://namenode2/foo
```

The `-delete` flag causes distcp to delete any files or directories from the destination that are not present in the source, and `-p` means that file status attributes like permissions, block size, and replication are preserved. You can run distcp with no arguments to see precise usage instructions.

If the two clusters are running incompatible versions of HDFS, then you can use the webhdfs protocol to distcp between them:

```
% hadoop distcp webhdfs://namenode1:50070/foo webhdfs://namenode2:50070/foo
```

Another variant is to use an HttpFs proxy as the distcp source or destination (again using the webhdfs protocol), which has the advantage of being able to set firewall and bandwidth controls (see HTTP).

4. Summarize the HDFS application.

Ans:

Hadoop HDFS Feature

a. Distributed Storage

As HDFS stores data in a distributed manner. It divides the data into small pieces and stores it in different nodes of the cluster. In this manner, Hadoop Distributed File System provides a way to map reduce to process a subset of large data, which is broken into smaller pieces and stored in multiple nodes, parallelly on several nodes. MapReduce is the heart of Hadoop but HDFS is the one which provides it all these capabilities.

b. Blocks

As HDFS splits huge files into small chunks known as blocks. Block is the smallest unit of data in a filesystem. We (client and admin) do not have any control on the block like block location. Namenode decides all such things. HDFS default block size is 128 MB which can be increased as per the requirement. This is unlike OS filesystem where the block size is 4 KB.

If the data size is less than the block size of HDFS, then block size will be equal to the data size. For example, if the file size is 129 MB, then 2 blocks will be created for it. One block will be of default size 128 MB and other will be 1 MB only and not 128 MB as it will waste the space (here block size is equal to data size). Hadoop is intelligent enough not to waste rest of 127 MB. So it is allocating 1 MB block only for 1 MB data.

The major advantage of storing data in such block size is that it saves disk seek time and another advantage is in the case of processing as **mapper** processes 1 block at a time. So 1 mapper processes large data at a time.

In Hadoop Distributed file system the file is split into blocks and each block is stored at different nodes with default 3 replicas of each block. Each replica of a block is stored at the different node to provide fault tolerant feature and the placement of these blocks on the different node is decided by Name node. Name node makes it as much distributed as possible. While placing a block on a data node, it considers how much a particular data node is loaded at that time.

c. Replication

Hadoop HDFS creates duplicate copies of each block. This is called replication. All blocks are replicated and stored at different nodes across the cluster. It tries to put at least 1 replica in every rack.

What do you mean by rack?

Datanodes are arranged in racks. All the nodes in a rack are connected by a single switch so if a switch or complete rack is down, data can be accessed from another rack. The default replication factor is 3 and this can be changed to the required values according to the requirement by editing the configuration files (hdfs-site.xml)

d. High Availability

Replication of data blocks and storing at multiple nodes across cluster provides high availability of data. Even if a network link or node or some hardware goes down, we can easily get the data from the different path or different node as data is minimally replicated at 3 nodes. This is how high availability feature is supported by HDFS.

e. Data Reliability

The data is replicated in HDFS, It is stored reliably as well. Due to replication, blocks are highly available even if some node crashes or some hardware fails. We can balance the cluster (if 1 node goes down, block that was stored at that node become under replicated or if a node which has gone down suddenly becomes active, block at that node is over replicated. We need to balance the cluster to create or destroy the replica as per the situation) in such case by making the replication factor to desired value by just running few commands. This is how data is stored reliably and provides fault tolerant and high availability.

f. Fault Tolerant

HDFS provides fault tolerant storage layer for Hadoop and other components in the ecosystem. HDFS works with commodity hardware (systems with average configurations) that has high chances of getting crashed any time. Thus, to make the entire system highly fault-tolerant, HDFS replicates and stores data in different places. Any data gets stored at 3 different locations by default. So, even if one of them is corrupted and the other is unavailable for some time for any reason, then data can be accessed from the third one. Hence, there is no chance of

losing the data. This replication factor helps us to attain the feature of Hadoop called Fault Tolerant.

g. Scalability

Scalability means expanding or contracting the cluster. In Hadoop HDFS, scalability is done in 2 ways.

a) We can add more disks on nodes of the cluster.

For doing this, we need to edit the configuration files and make corresponding entries of newly added disks. Here we need to provide down time though it is very less. So people generally prefer second way of scaling which is horizontal scaling.

b) Another option of scalability is of adding more nodes to the cluster on the fly without any downtime. This is known as horizontal scaling.

We can add as many nodes as we want in the cluster on the fly in real time without any downtime. This is a unique feature provided by Hadoop.

h. High throughput access to application data

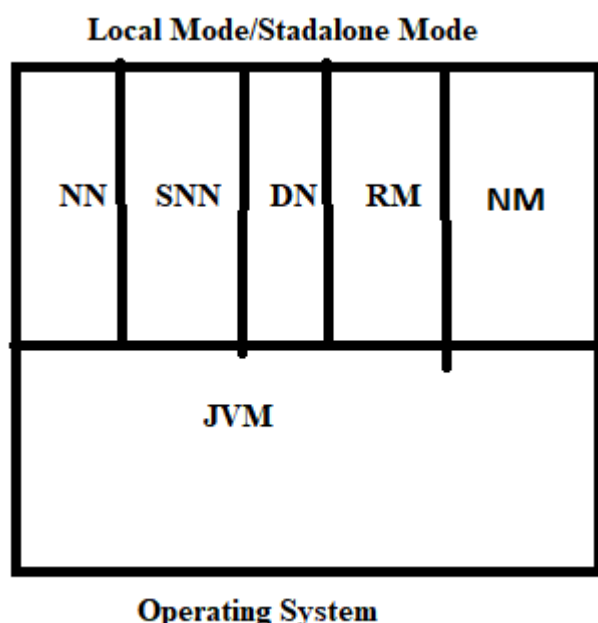
Hadoop Distributed File System provides high throughput access to application data. Throughput is the amount of work done in a unit time. It describes how fast the data is getting accessed from the system and it is usually used to measure the performance of the system. When we want to perform a task or an action, then the work is divided and shared among different systems. So all the systems will be executing the tasks assigned to them independently and in parallel. So the work will be completed in a very short period of time. In this way, the HDFS gives good throughput. By reading data in parallel, we decrease the actual time to read data tremendously.

5. Discuss the installation of HADOOP in three operating modes.

Ans:

we can install hadoop in 3 ways.

1. Local Mode/ Standalone Mode
2. pSudo Distribution Mode
3. Fully Distribution Mode.

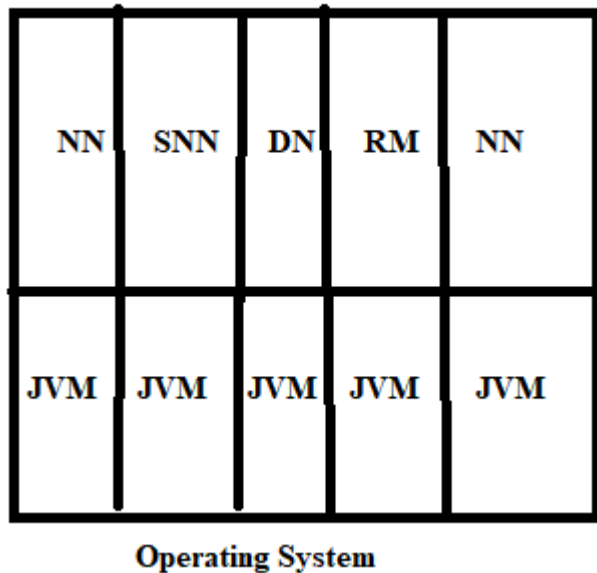


Local Mode/ Standalone Mode

=====

- > All daemons runs same machine on single jvm
- > These is not distributed concept
- > It uses local file system
- > We use this local mode for development
- > It is useful for application debugging
- > It is not used for production.

pSudo Distribution Mode

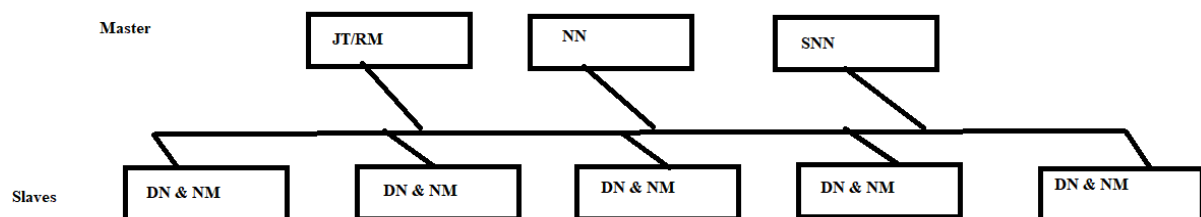


psudo Distribution Mode

=====

- > All daemons runs same machine but on separate jvm
- > These is distributed concept
- > It uses HDFS file system
- > It is used for development, testing but not for production.

Fully Distribution



Fully Distribution Mode

=====

- > It is recommended to run each master Daemon on separate machine
- > It is highly recommended to run both slave daemons on same machine.
- > It is used for development, testing and production

6. Illustrate various HDFS commands.

Ans:

HDFS Commands: Hadoop Shell Commands to Manage HDFS

Important HDFS commands and their working which are used most frequently when working with Hadoop File System.

- fsck

HDFS Command to check the health of the Hadoop file system.

Command: `hdfs fsck /`

- ls

HDFS Command to display the list of Files and Directories in HDFS.

Command: `hdfs dfs -ls /`

- mkdir

HDFS Command to create the directory in HDFS.

Usage: `hdfs dfs -mkdir /directory_name`

Command: `hdfs dfs -mkdir /lbrce`

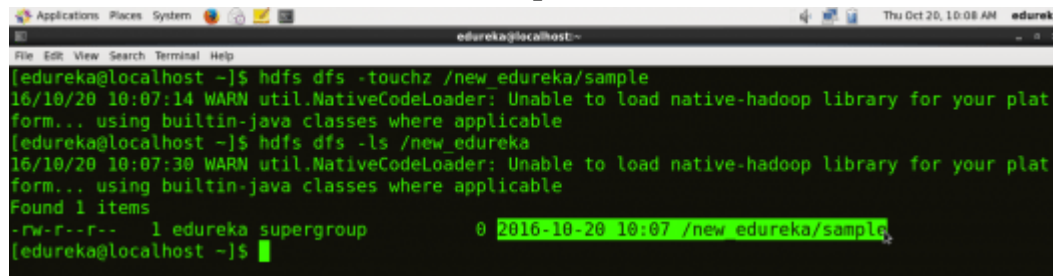
Note: Here we are trying to create a directory named “lbrce” in HDFS.

- touchz

HDFS Command to create a file in HDFS with file size 0 bytes.

Usage: `hdfs dfs -touchz /directory/filename`

Command: `hdfs dfs -touchz /lbrce/sample`

A screenshot of a terminal window titled 'edureka@localhost:~'. The terminal shows the following commands and output:

```
[edureka@localhost ~]$ hdfs dfs -touchz /new_edureka/sample
16/10/20 10:07:14 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your plat
form... using builtin-java classes where applicable
[edureka@localhost ~]$ hdfs dfs -ls /new_edureka
16/10/20 10:07:30 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your plat
form... using builtin-java classes where applicable
Found 1 items
-rw-r--r-- 1 edureka supergroup 0 2016-10-20 10:07 /new_edureka/sample
[edureka@localhost ~]$
```

Note: Here we are trying to create a file named “sample” in the directory “lbrce” of hdfs with file size 0 bytes.

- du

HDFS Command to check the file size.

Usage: `hdfs dfs -du -s /directory/filename`

Command: `hdfs dfs -du -s /lbrce/sample`

- cat

HDFS Command that reads a file on HDFS and prints the content of that file to the standard output.

Usage: `hdfs dfs -cat /path/to/file_in_hdfs`

Command: `hdfs dfs -cat /lbrce/test`

- text

HDFS Command that takes a source file and outputs the file in text format.

Usage: `hdfs dfs -text /directory/filename`

Command: `hdfs dfs -text /lbrce/test`

- copyFromLocal

HDFS Command to copy the file from a Local file system to HDFS.

Usage: `hdfs dfs -copyFromLocal <localsrc> <hdfs destination>`

Command: `hdfs dfs -copyFromLocal /home/lbrce/test /lbrce`

Note: Here the test is the file present in the local directory /home/lbrce and after the command gets executed the test file will be copied in /lbrce directory of HDFS.

- copyToLocal

HDFS Command to copy the file from HDFS to Local File System.

Usage: `hdfs dfs -copyToLocal <hdfs source> <localdst>`

Command: `hdfs dfs -copyToLocal /lbrce/test /home/lbrce`

Note: Here test is a file present in the lbrce directory of HDFS and after the command gets executed the test file will be copied to local directory /home/lbrce

- put

HDFS Command to copy single source or multiple sources from local file system to the destination file system.

Usage: `hdfs dfs -put <localsrc> <destination>`

Command: `hdfs dfs -put /home/ lbrce /test /user`

Note: The command copyFromLocal is similar to put command, except that the source is restricted to a local file reference.

- get

HDFS Command to copy files from hdfs to the local file system.

Usage: `hdfs dfs -get <src> <localdst>`

Command: `hdfs dfs -get /user/test /home/ lbrce`

Note: The command copyToLocal is similar to get command, except that the destination is restricted to a local file reference.

- count

HDFS Command to count the number of directories, files, and bytes under the paths that match the specified file pattern.

Usage: `hdfs dfs -count <path>`

Command: `hdfs dfs -count /user`

- rm

HDFS Command to remove the file from HDFS.

Usage: `hdfs dfs -rm <path>`

Command: `hdfs dfs -rm / lbrce /test`

- rm -r

HDFS Command to remove the entire directory and all of its content from HDFS.

Usage: `hdfs dfs -rm -r <path>`

Command: `hdfs dfs -rm -r / lbrce`

- cp

HDFS Command to copy files from source to destination. This command allows multiple sources as well, in which case the destination must be a directory.

Usage: `hdfs dfs -cp <src> <dest>`

Command: `hdfs dfs -cp /user/hadoop/file1 /user/hadoop/file2`

Command: `hdfs dfs -cp /user/hadoop/file1 /user/hadoop/file2 /user/hadoop/dir`

- mv

HDFS Command to move files from source to destination. This command allows multiple sources as well, in which case the destination needs to be a directory.

Usage: `hdfs dfs -mv <src> <dest>`

Command: `hdfs dfs -mv /user/hadoop/file1 /user/hadoop/file2`

- expunge

HDFS Command that makes the trash empty.

Command: `hdfs dfs -expunge`

- rmdir

HDFS Command to remove the directory.

Usage: `hdfs dfs -rmdir <path>`

Command: `hdfs dfs -rmdir /user/hadoop`

- usage

HDFS Command that returns the help for an individual command.

Usage: `hdfs dfs -usage <command>`

Command: `hdfs dfs -usage mkdir`

Note: By using usage command you can get information about any command.

- help

HDFS Command that displays help for given command or all commands if none is specified.

Command: `hdfs dfs -help`

7. Describe the Hadoop applications.

Ans:

Features of HDFS

- **Cost:** The HDFS, in general, is deployed on a commodity hardware like your desktop/laptop which you use every day. So, it is very economical in terms of the cost of ownership of the project. Since, we are using low cost commodity hardware, you don't need to spend huge amount of money for scaling out your Hadoop cluster. In other words, adding more nodes to your HDFS is cost effective.
- **Variety and Volume of Data:** When we talk about HDFS then we talk about storing huge data i.e. Terabytes & petabytes of data and different kinds of data. So, you can store any type of data into HDFS, be it structured, unstructured or semi structured.
- **Reliability and Fault Tolerance:** When you store data on HDFS, it internally divides the given data into data blocks and stores it in a distributed fashion across your Hadoop cluster. The information regarding which data block is located on which of the data nodes is recorded in the metadata. **NameNode** manages the meta data and the **DataNodes** are responsible for storing the data. Name node also replicates the data i.e. maintains multiple copies of the data. This replication of the data makes HDFS very reliable and fault tolerant. So, even if any of the nodes fails, we can retrieve the data from the replicas residing on other data nodes. By default, the replication factor is 3. Therefore, if you store 1 GB of file in HDFS, it will finally occupy 3 GB of space. The name node periodically updates the metadata and maintains the replication factor consistent.
- **Data Integrity:** Data Integrity talks about whether the data stored in my HDFS are correct or not. HDFS constantly checks the integrity of data stored against its checksum. If it finds any fault, it reports to the name node about it. Then, the name node creates additional new replicas and therefore deletes the corrupted copies.
- **High Throughput:** Throughput is the amount of work done in a unit time. It talks about how fast you can access the data from the file system. Basically, it gives you an insight about the system performance. As you have seen in the above example where we used ten machines collectively to enhance computation. There we were able to reduce the processing time from **43 minutes** to a mere **4.3 minutes** as all the machines were working in parallel. Therefore, by processing data in parallel, we decreased the processing time tremendously and thus, achieved high throughput.

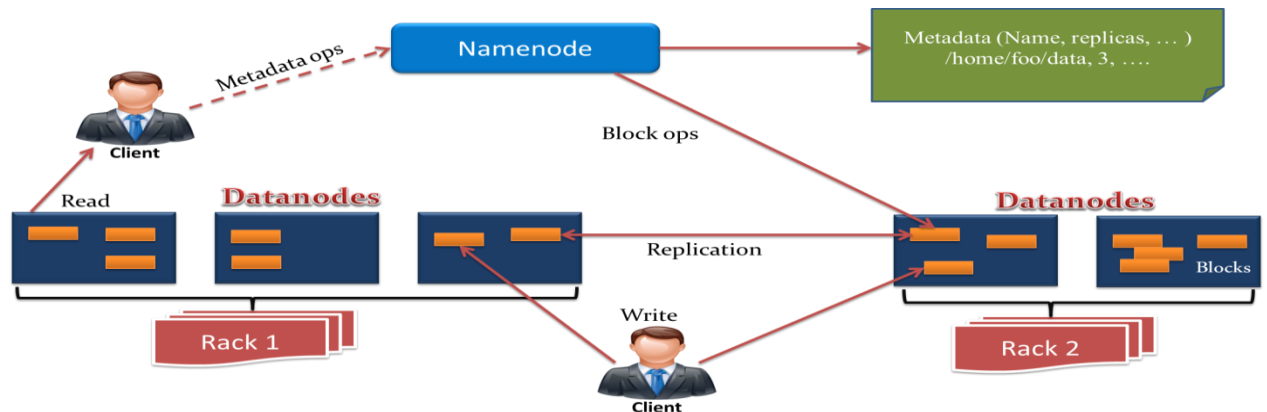
Data Locality: Data locality talks about moving processing unit to data rather than the data to processing unit. In our traditional system, we used to bring the data to the application layer and then process it. But now, because of the architecture and huge volume of the data, bringing the data to the application layer will reduce the network performance to a noticeable extent. So, in HDFS, we bring the computation part to the data nodes where the data is residing. Hence, you are not moving the data, you are bringing the program or processing part to the data.

8. Summarize the HDFS design.

Ans:

HDFS Architecture

This architecture gives you a complete picture of Hadoop Distributed File System. There is a single namenode which stores metadata and there are multiple datanodes which do actual storage work. Nodes are arranged in racks and Replicas of data blocks are stored on different racks in the cluster to provide fault tolerance. To read or write a file in HDFS, the client needs to interact with Namenode. HDFS applications need a *write-once-read-many* access model for files. A file once created and written cannot be edited.



Namenode stores metadata and datanode which stores actual data. The client interacts with namenode for any task to be performed as namenode is the centerpiece in the cluster. There are several datanodes in the cluster which store HDFS data in the local disk. Datanode sends a heartbeat message to namenode periodically to indicate that it is alive. Also, it replicates data to other datanode as per the replication factor.

9. List and explain various Hadoop data types.

Ans:

the list of few data types in Java along with the equivalent Hadoop variant:

1. **Integer** → **IntWritable**: It is the Hadoop variant of *Integer*. It is used to pass integer numbers as key or value.
2. **Float** → **FloatWritable**: Hadoop variant of *Float* used to pass floating point numbers as key or value.
3. **Long** → **LongWritable**: Hadoop variant of *Long* data type to store long values.
4. **Short** → **ShortWritable**: Hadoop variant of *Short* data type to store short values.
5. **Double** → **DoubleWritable**: Hadoop variant of *Double* to store double values.
6. **String** → **Text**: Hadoop variant of *String* to pass string characters as key or value.
7. **Byte** → **ByteWritable**: Hadoop variant of *byte* to store sequence of bytes.
8. **null** → **NullWritable**: Hadoop variant of *null* to pass null as a key or value.

Usually *NullWritable* is used as data type for output key of the reducer, when the output key is not important in the final result.

10. List and explain the Features of MapReduce.

Ans:

Advantages of MapReduce here two biggest advantages of MapReduce are:

1. Parallel Processing:

In MapReduce, we are dividing the job among multiple nodes and each node works with a part of the job simultaneously. So, MapReduce is based on Divide and Conquer paradigm which

helps us to process the data using different machines. As the data is processed by multiple machine instead of a single machine in parallel, the time taken to process the data gets reduced by a tremendous amount as shown in the figure below (2).

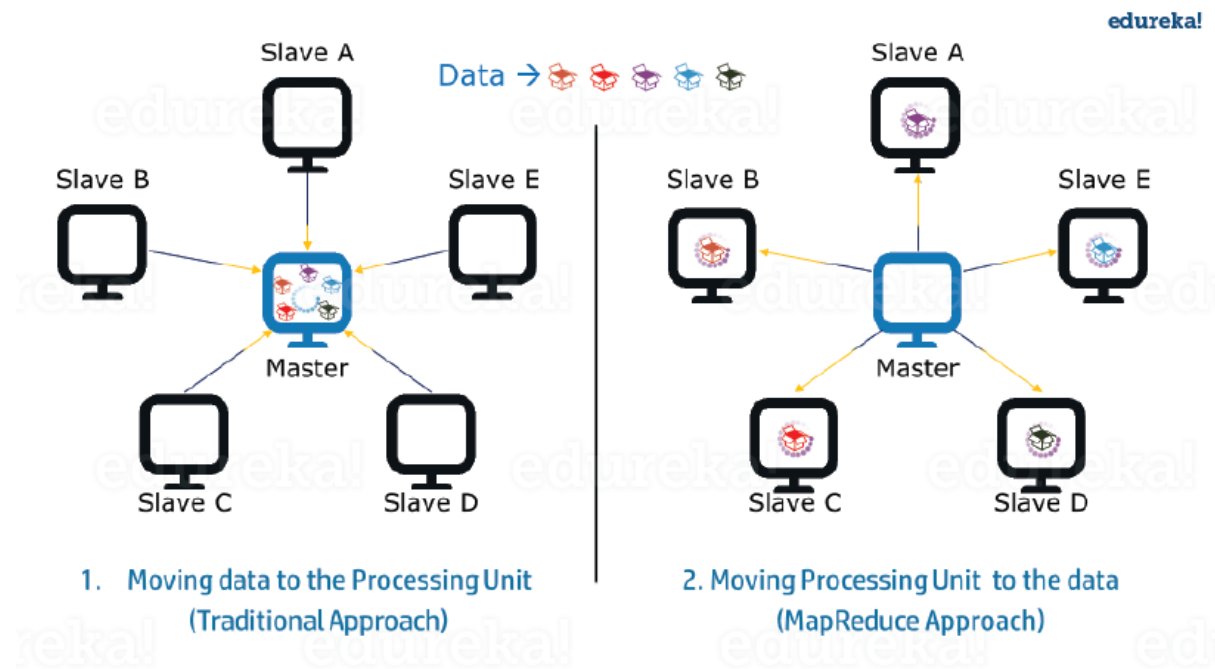


Fig.: Traditional Way Vs. MapReduce Way – MapReduce

2. Data Locality:

Instead of moving data to the processing unit, we are moving processing unit to the data in the MapReduce Framework. In the traditional system, we used to bring data to the processing unit and process it. But, as the data grew and became very huge, bringing this huge amount of data to the processing unit posed following issues:

- Moving huge data to processing is costly and deteriorates the network performance.
- Processing takes time as the data is processed by a single unit which becomes the bottleneck.
- Master node can get over-burdened and may fail.

Now, MapReduce allows us to overcome above issues by bringing the processing unit to the data. So, as you can see in the above image that the data is distributed among multiple nodes where each node processes the part of the data residing on it. This allows us to have the following advantages:

- It is very cost effective to move processing unit to the data.
- The processing time is reduced as all the nodes are working with their part of the data in parallel.
- Every node gets a part of the data to process and therefore, there is no chance of a node getting overburdened.

11. Implement Word count MapReduce program.

Ans:

```
package com.cfamilycomputers.wordcount;
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
public class WordCountMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
    @Override
```



```

    public void map(LongWritable key, Text value, Context con)
        throws IOException, InterruptedException {
        String line = value.toString();
        String[] words = line.split("\\s");
        for(String s:words) {
            con.write(new Text(s), new IntWritable(1));
        }
    }
}

package com.cfamilychcomputers.wordcount;
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
public class WordCountReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
    @Override
    public void reduce(Text key, Iterable<IntWritable> values,Context con)
        throws IOException, InterruptedException {
        int sum = 0;
        for(IntWritable i:values) {
            sum = sum + i.get();
        }
        con.write(key, new IntWritable(sum));
    }
}

package com.cfamilychcomputers.wordcount;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
public class WordCountDriver {
    public static void main(String[] args) throws Exception {
        if (args.length != 2) {
            System.err.println("Usage: WordCount <input path> <output path>");
            System.exit(-1);
        }
        Job job = new Job();
        job.setJarByClass(WordCountDriver.class);
        job.setJobName("Word Count");
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        job.setMapperClass(WordCountMapper.class);
        job.setReducerClass(WordCountReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}

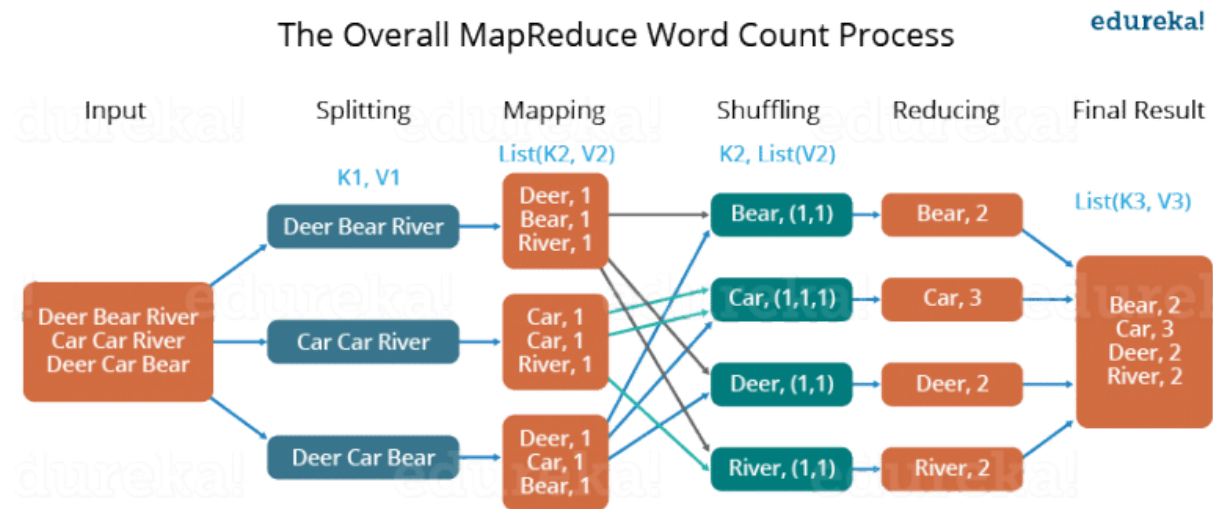
```

12. Explain the MapReduce Word count problem procedure with different phases.

Ans:

A Word Count Example of MapReduce

We have to perform a word count on the sample.txt using MapReduce. So, we will be finding the unique words and the number of occurrences of those unique words.



- ☐ First, we divide the input in three splits as shown in the figure. This will distribute the work among all the map nodes.
- ☐ Then, we tokenize the words in each of the mapper and give a hardcoded value (1) to each of the tokens or words. The rationale behind giving a hardcoded value equal to 1 is that every word, in itself, will occur once.
- ☐ Now, a list of key-value pair will be created where the key is nothing but the individual words and value is one. So, for the first line (Dear Bear River) we have 3 key-value pairs – Deer, 1; Bear, 1; River, 1. The mapping process remains the same on all the nodes.
- ☐ After mapper phase, a partition process takes place where sorting and shuffling happens so that all the tuples with the same key are sent to the corresponding reducer.
- ☐ So, after the sorting and shuffling phase, each reducer will have a unique key and a list of values corresponding to that very key. For example, Bear, [1,1]; Car, [1,1,1].., etc.
- ☐ Now, each Reducer counts the values which are present in that list of values. As shown in the figure, reducer gets a list of values which is [1,1] for the key Bear. Then, it counts the number of ones in the very list and gives the final output as – Bear, 2.
- ☐ Finally, all the output key/value pairs are then collected and written in the output file.

13. Explain about Java Interface for HDFS File I/O.

To explore more into Hadoop distributed file system through Java Interface, we must have knowledge on a few important main classes which provide I/O operations on Hadoop files.

FileSystem – org.apache.hadoop.fs – An abstract file system API.

IOUtils – org.apache.hadoop.io – Generic i/o code for reading and writing data to HDFS.

IOUtils:

It is a utility class (handy tool) for I/O related functionality on HDFS. It is present in **org.apache.hadoop.io** package.

Below are some of its important methods which we use very frequently in HDFS File I/O Operations. All these methods are static methods.

copyBytes:

IOUtils.copyBytes(InputStream in, OutputStream out, int buffSize, boolean close) ;
This method copies data from one stream to another. The last two arguments are the buffer size used for copying and whether to close the streams when the copy is complete.

closeStream:

IOUtils.closeStream(Closeable stream) ;
This method is used to close input or output streams irrespective of any IOException. This method is generally placed finally clause of a try-catch block.

Configuration:

This class provides access to configuration parameters on a client or server machine. This class is present in **org.apache.hadoop.conf** package.

Configurations are specified by resources/properties. A property contains a set of name/value pairs as XML data. Each resource is named by a String.

By default hadoop loads configuration parameters from two files.

1. core-default.xml – default configuration properties
2. conf/core-site.xml – site specific configuration properties

For example, properties are defined in the above two files as shown below.

```
<property>
  <name>fs.defaultFS</name>
  <value>hdfs://localhost:9000</value>
</property>
```

Applications can add additional properties on Configuration object.

Below are some of the useful methods on configuration object.

Configuration conf = new Configuration() — default configuration parameters will be returned.

Addition of Resources:

conf.addResource(String name) — adds a resource called 'name'

Getting Property Values:

conf.get(String name) — gets value of the property 'name'.

conf.getBoolean(String name, boolean defaultValue) — gets value of property 'name' as boolean

conf.getClass(String name, Class<?> defaultValue) — gets class of property 'name'

conf.getDouble(String name, double defaultValue)

conf.getFloat(String name, float defaultValue)

conf.getInt(String name, int defaultValue)

conf.getStrings(String name) — Gets ',' delimited values of 'name' property as an array of strings.

Setting Property Values:

conf.set(String name, String value) — Set the value of the name property.

conf.setBoolean(String name, boolean value) — set the 'name' property to a boolean 'value'.

conf.setClass(String name, Class<?> theClass, Class<?> interface) — Set the value of the 'name' property to the name of a *theClass* implementing the given interface *'interface'*

conf.setDouble(String name, double value)

conf.setEnum(String name, T value)

conf.setFloat(String name, float value)

conf.setInt(String name, int value)

FileSystem:

FileSystem is an abstract base class for a generic file system. It may be implemented as distributed system or a local system. The local implementation is **LocalFileSystem** and distributed implementation is **DistributedFileSystem**.

All these classes are present in **org.apache.hadoop.fs** package.

All user code that may use the HDFS should use a **FileSystem** object.

Similar to **DataInputStream** and **DataOutputStream** in Java File I/O for reading or writing primitive data types, In Hadoop, their corresponding stream classes are **FSDaataInputStream** and **FSDaataOutputStream** respectively.

FSDaataInputStream:

- **FSDaataInputStream** class is a specialization of *java.io.DataInputStream* with support for random access, so we can read from any part of the stream. It is an utility that wraps a *FSInputStream* in a *DataInputStream* and buffers input through a *BufferedInputStream*.
- *FSDaataInputStream* class implements **Seekable & PositionedReadable** interfaces. So, we can have random access in the stream with help of below methods.

`int read(long position, byte[] buffer, int offset, int length)` – Read bytes from the given position in the stream to the given buffer. The return value is the number of bytes actually read.

`void readFully(long position, byte[] buffer, int offset, int length)` – Read bytes from the given position in the stream to the given buffer. Continues to read until length bytes have been read. If the end of stream is reached while reading *EOFException* is thrown.

`void readFully(long position, byte[] buffer)` – `buffer.length` bytes will be read from position in stream

`void seek(long pos)` – Seek to the given offset.

`long getPos()` – Get the current position in the input stream.

FSDaataOutputStream:

- *FSDaataOutputStream* class is counterpart for *FSDaataInputStream*, to open a stream for output. It is an utility that wraps a *OutputStream* in a *DataOutputStream*, buffers output through a *BufferedOutputStream* and creates a checksum file.
- Similar to *FSDaataInputStream*, *FSDaataOutputStream* also **support** `getPos()` method to know current position in the output stream but **seek()** method **is not** supported by *FSDaataOutputStream*.

It is because HDFS allows only sequential writes to an open file or appends to an existing File. In other words, there is no support for writing to anywhere other than the end of the file.

- We can invoke **write()** method to write to an output stream on an instance of *FSDaataOutputStream*.

`public void write(byte[] b, int off, int len)` throws *IOException* ;

Writes **len** bytes from the specified byte array starting at offset **off** to the underlying output stream. If no exception is thrown, the counter written is incremented by **len**.

Below are some of the important methods from FileSystem class.

Getting FileSystem Instance:

For any File I/O operation in HDFS through Java API, the first thing we need is *FileSystem* instance. To get file system instance, we have three static methods from *FileSystem* class.

- static *FileSystem* `get(Configuration conf)` — Returns the configured file system implementation.
- static *FileSystem* `get(URI uri, Configuration conf)` — Returns the *FileSystem* for this URI.

- static `FileSystem get(Uri uri, Configuration conf, String user)` — Get a file system instance based on the uri, the passed configuration and the user.

Opening Existing File:

In order to read a file from HDFS, we need to open an input stream for the same. We can do the same by invoking **open()** method on `FileSystem` instance.

- `public FSDataInputStream open(Path f)`
- `public abstract FSDataInputStream open(Path f, int bufferSize)`

The first method uses a default buffer size of 4 K.

Creating a new File:

There are several ways to create a file in HDFS through **FileSystem** class. But one of the simplest method is to invoke **create()** method that takes a **Path** object for the file to be created and returns an output stream to write to.

```
public FSDataOutputStream create(Path f)
```

There are **overloaded versions of this method** that **allow** you to specify whether to **forcibly overwrite existing files**, the **replication factor of the file**, the buffer size to use when writing the file, the block size for the file, and file permissions.

The `create()` method **creates any parent directories of the file to be written that don't already exist**.

Below is a sample program for reading data from HDFS file to Standard output.

```
import java.io.*;
import java.net.URI;
import org.apache.hadoop.fs.*;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;

public class HdfsRead {
    public static void main(String[] args) throws IOException {

        String uri = args[0];

        Configuration conf = new Configuration();
        FileSystem fs = FileSystem.get(URI.create(uri), conf);
        FSDataInputStream in = null ;

        try
        {
            in = fs.open(new Path(uri));
            IOUtils.copyBytes(in, System.out, 4096, false);
        }
        finally
        {
            IOUtils.closeStream(in);
        }
    }
}
```

Below is a sample program for copying data into a HDFS file from another HDFS file.

```
import java.io.*;
import java.net.URI;
import org.apache.hadoop.fs.*;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;

public class HdfsRead {
    public static void main(String[] args) throws IOException {

        String uri = args[0];

        Configuration conf = new Configuration();
        FileSystem fs = FileSystem.get(URI.create(uri), conf);
        FSDataInputStream in = null ;

        try
        {
            in = fs.open(new Path(uri));
            IOUtils.copyBytes(in, System.out, 4096, false);
        }
        finally
        {
            IOUtils.closeStream(in);
        }
    }
}
```

14. Explain about k-means Clustering.

K-Means clustering is an unsupervised learning algorithm. There is no labeled data for this clustering, unlike in supervised learning. K-Means performs division of objects into clusters that share similarities and are dissimilar to the objects belonging to another cluster.

The term 'K' is a number. You need to tell the system how many clusters you need to create. For example, K = 2 refers to two clusters. There is a way of finding out what is the best or optimum value of K for a given data.

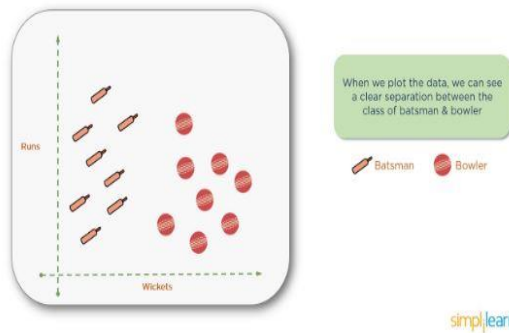
For a better understanding of k-means, let's take an example from cricket. Imagine you received data on a lot of cricket players from all over the world, which gives information on the runs scored by the player and the wickets taken by them in the last ten matches. Based on this information, we need to group the data into two clusters, namely batsman and bowlers. Let's take a look at the steps to create these clusters.

Solution:

Assign data points

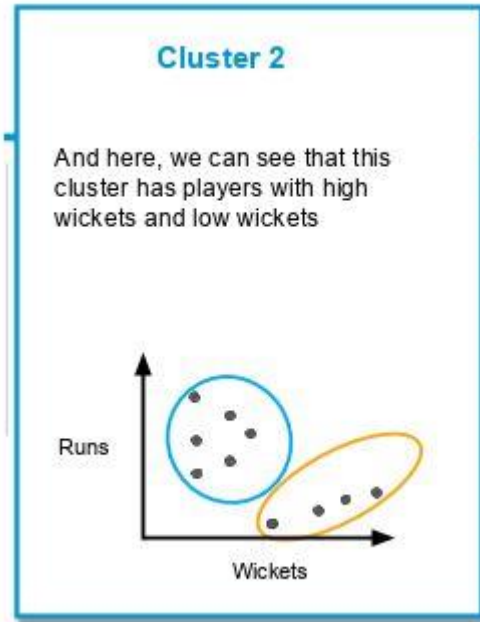
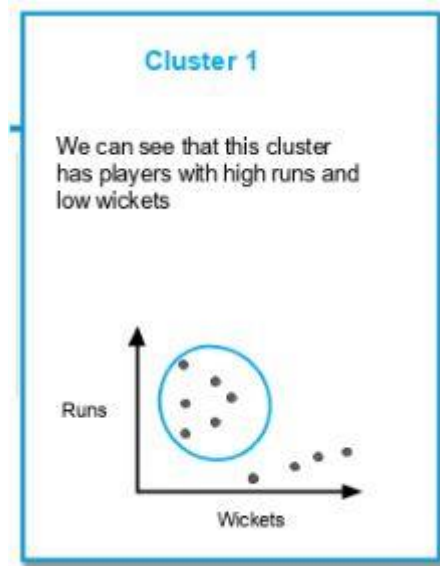
Here, we have our data set plotted on 'x' and 'y' coordinates. The information on the y-axis is about the runs scored, and on the x-axis about the wickets taken by the players.

If we plot the data, this is how it would look:



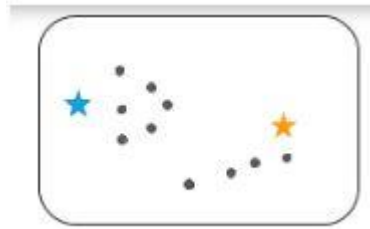
Perform Clustering

We need to create the clusters, as shown below:



Considering the same data set, let us solve the problem using K-Means clustering (taking $K = 2$).

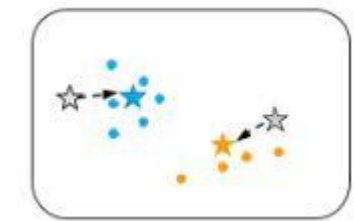
The first step in k-means clustering is the allocation of two centroids randomly (as $K=2$). Two points are assigned as centroids. Note that the points can be anywhere, as they are random points. They are called centroids, but initially, they are not the central point of a given data set.



The next step is to determine the distance between each of the data points from the randomly assigned centroids. For every point, the distance is measured from both the centroids, and whichever distance is less, that point is assigned to that centroid. You can see the data points attached to the centroids and represented here in blue and yellow.



The next step is to determine the actual centroid for these two clusters. The original randomly allocated centroid is to be repositioned to the actual centroid of the clusters.



This process of calculating the distance and repositioning the centroid continues until we obtain our final cluster. Then the centroid repositioning stops.



As seen above, the centroid doesn't need anymore repositioning, and it means the algorithm has converged, and we have the two clusters with a centroid.

Applications of K-Means Clustering

K-Means clustering is used in a variety of examples or business cases in real life, like:

- Academic performance
- Diagnostic systems
- Search engines
- Wireless sensor networks

Academic Performance

Based on the scores, students are categorized into grades like A, B, or C.

Diagnostic systems

The medical profession uses k-means in creating smarter medical decision support systems, especially in the treatment of liver ailments.

Search engines

Clustering forms a backbone of search engines. When a search is performed, the search results need to be grouped, and the search engines very often use clustering to do this.

Wireless sensor networks

The clustering algorithm plays the role of finding the cluster heads, which collects all the data in its respective cluster.

Distance Measure

Distance measure determines the similarity between two elements and influences the shape of clusters.

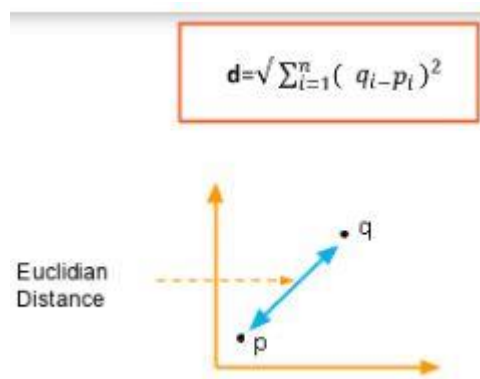
K-Means clustering supports various kinds of distance measures, such as:

- Euclidean distance measure
- Manhattan distance measure
- A squared euclidean distance measure
- Cosine distance measure

Euclidean Distance Measure

The most common case is determining the distance between two points. If we have a point P and point Q, the euclidean distance is an ordinary straight line. It is the distance between the two points in Euclidean space.

The formula for distance between two points is shown below:

$$d = \sqrt{\sum_{i=1}^n (q_i - p_i)^2}$$


Squared Euclidean Distance Measure

This is identical to the Euclidean distance measurement but does not take the square root at the end. The formula is shown below:

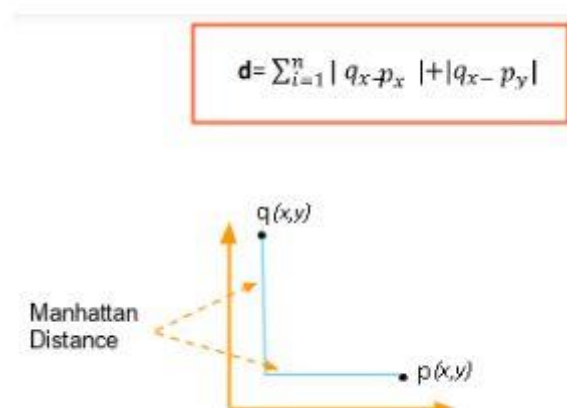
$$d = \sum_{i=1}^n (q_i - p_i)^2$$

Manhattan Distance Measure

The Manhattan distance is the simple sum of the horizontal and vertical components or the distance between two points measured along axes at right angles.

Note that we are taking the absolute value so that the negative values don't come into play.

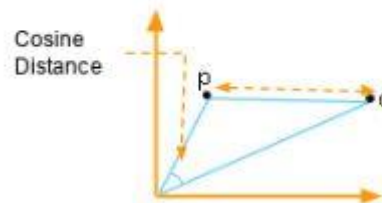
The formula is shown below:

$$d = \sum_{i=1}^n |q_x - p_x| + |q_y - p_y|$$


Cosine Distance Measure

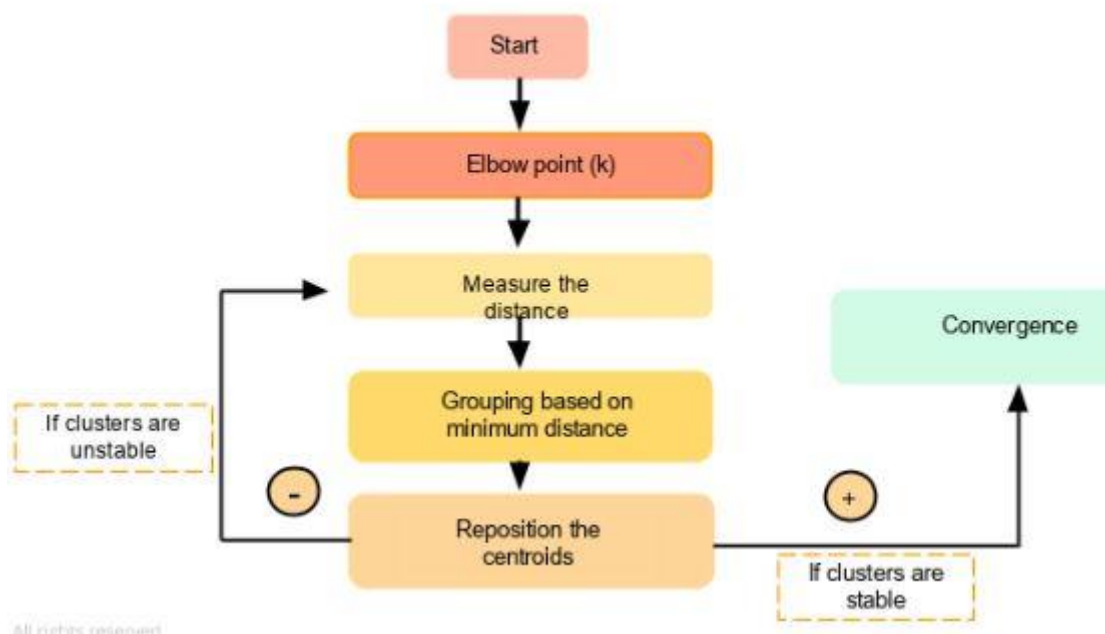
In this case, we take the angle between the two vectors formed by joining the points from the origin. The formula is shown below:

$$d = \frac{\sum_{i=0}^{n-1} q_i \cdot p_i}{\sqrt{\sum_{i=0}^{n-1} (q_i)^2 \times \sum_{i=0}^{n-1} (p_i)^2}}$$



How Does K-Means Clustering Work?

The flowchart below shows how k-means clustering works:



The goal of the K-Means algorithm is to find clusters in the given input data. There are a couple of ways to accomplish this. We can use the trial and error method by specifying the value of K (e.g., 3,4, 5). As we progress, we keep changing the value until we get the best clusters.

Another method is to use the Elbow technique to determine the value of K. Once we get the value of K, the system will assign that many centroids randomly and measure the distance of each of the data points from these centroids. Accordingly, it assigns those points to the corresponding centroid from which the distance is minimum. So each data point will be assigned to the centroid, which is closest to it. Thereby we have a K number of initial clusters.

For the newly formed clusters, it calculates the new centroid position. The position of the centroid moves compared to the randomly allocated one.

Once again, the distance of each point is measured from this new centroid point. If required, the data points are relocated to the new centroids, and the mean position or the new centroid is calculated once again.

If the centroid moves, the iteration continues indicating no convergence. But once the centroid stops moving (which means that the clustering process has converged), it will reflect the result.

Let's use a visualization example to understand this better.

We have a data set for a grocery shop, and we want to find out how many clusters this has to be spread across. To find the optimum number of clusters, we break it down into the following steps:

Step 1:

The Elbow method is the best way to find the number of clusters. The elbow method constitutes running K-Means clustering on the dataset.

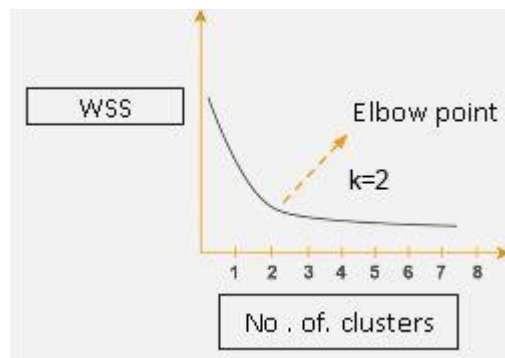
Next, we use within-sum-of-squares as a measure to find the optimum number of clusters that can be formed for a given data set. Within the sum of squares (WSS) is defined as the sum of the squared distance between each member of the cluster and its centroid.

$$WSS = \sum_{i=1}^m (x_i - c_i)^2$$

Where x_i = data point and c_i = closest point to centroid

The WSS is measured for each value of K. The value of K, which has the least amount of WSS, is taken as the optimum value.

Now, we draw a curve between WSS and the number of clusters.



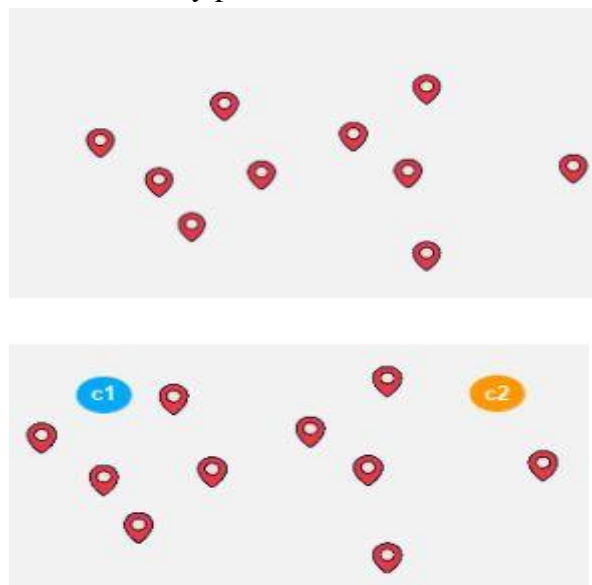
Here, WSS is on the y-axis and number of clusters on the x-axis.

You can see that there is a very gradual change in the value of WSS as the K value increases from 2.

So, you can take the elbow point value as the optimal value of K. It should be either two, three, or at most four. But, beyond that, increasing the number of clusters does not dramatically change the value in WSS, it gets stabilized.

Step 2:

Let's assume that these are our delivery points:



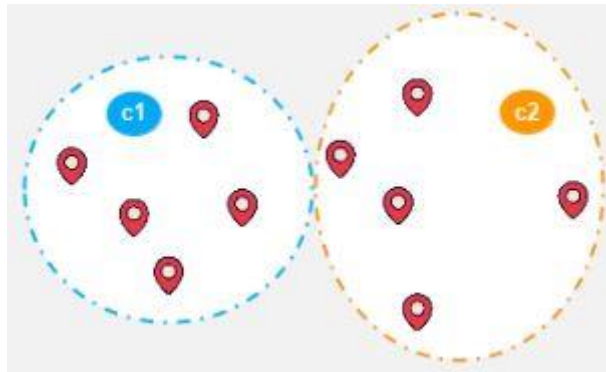
We can randomly initialize two points called the cluster centroids.

Here, C1 and C2 are the centroids assigned randomly.

Step 3:

Now the distance of each location from the centroid is measured, and each data point is assigned to the centroid, which is closest to it.

This is how the initial grouping is done:

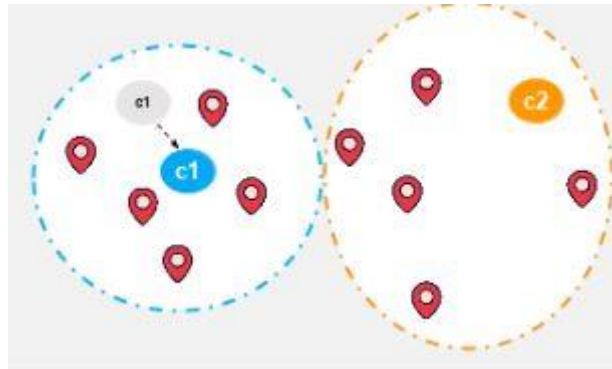


Step 4:

Compute the actual centroid of data points for the first group.

Step 5:

Reposition the random centroid to the actual centroid.

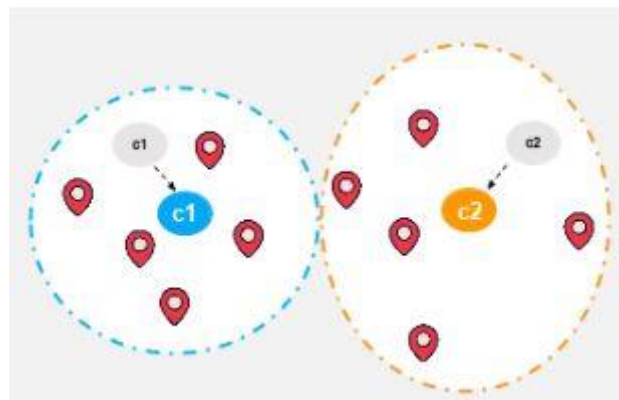


Step 6:

Compute the actual centroid of data points for the second group.

Step 7:

Reposition the random centroid to the actual centroid.



Step 8:

Once the cluster becomes static, the k-means algorithm is said to be converged.

The final cluster with centroids c1 and c2 is as shown below:



K-Means Clustering Algorithm

Let's say we have $x_1, x_2, x_3, \dots, x_n$ as our inputs, and we want to split this into K clusters.

The steps to form clusters are:

Step 1: Choose K random points as cluster centers called centroids.

Step 2: Assign each $x(i)$ to the closest cluster by implementing euclidean distance (i.e., calculating its distance to each centroid)

Step 3: Identify new centroids by taking the average of the assigned points.

Step 4: Keep repeating step 2 and step 3 until convergence is achieved

Let's take a detailed look at it at each of these steps.

Step 1:

We randomly pick K (centroids). We name them c_1, c_2, \dots, c_k , and we can say that

$$C = \{c_1, c_2, \dots, c_k\}$$

Where C is the set of all centroids.

Step 2:

We assign each data point to its nearest center, which is accomplished by calculating the euclidean distance.

$$\arg \min_{c_i \in C} \text{dist}(c_i, x)^2$$

Where $\text{dist}()$ is the Euclidean distance.

Here, we calculate the distance of each x value from each c value, i.e. the distance between $x_1-c_1, x_1-c_2, x_1-c_3$, and so on. Then we find which is the lowest value and assign x_1 to that particular centroid.

Similarly, we find the minimum distance for x_2, x_3 , etc.

Step 3:

We identify the actual centroid by taking the average of all the points assigned to that cluster.

$$c_i = \frac{1}{|S_i|} \sum_{x_j \in S_i} x_j$$

Where S_i is the set of all points assigned to the i th cluster.

It means the original point, which we thought was the centroid, will shift to the new position, which is the actual centroid for each of these groups.

Step 4:

Keep repeating step 2 and step 3 until convergence is achieved.

15. Describe the HDFS read and write operations.

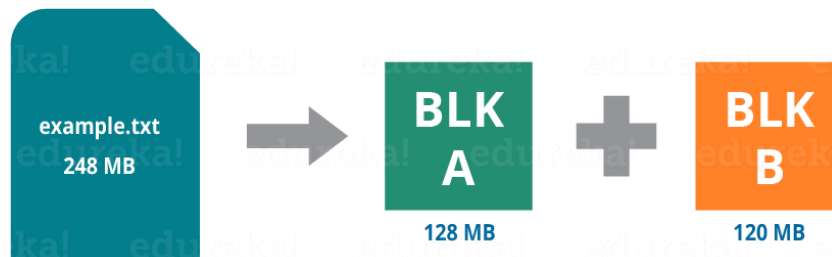
Ans:

HDFS Read/ Write Architecture:

Now let's talk about how the data read/write operations are performed on HDFS. HDFS follows Write Once – Read Many Philosophy. So, you can't edit files already stored in HDFS. But, you can append new data by re-opening the file.

HDFS Write Architecture:

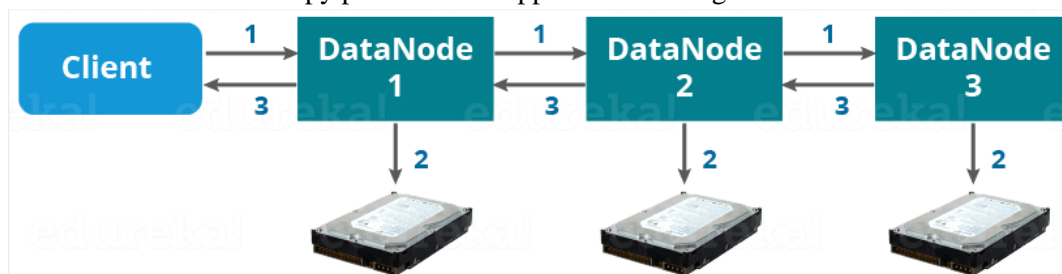
Suppose a situation where an HDFS client, wants to write a file named “example.txt” of size 248 MB.



Assume that the system block size is configured for 128 MB (default). So, the client will be dividing the file “example.txt” into 2 blocks – one of 128 MB (Block A) and the other of 120 MB (block B).

Now, the following protocol will be followed whenever the data is written into HDFS:

- At first, the HDFS client will reach out to the NameNode for a Write Request against the two blocks, say, Block A & Block B.
- The NameNode will then grant the client the write permission and will provide the IP addresses of the DataNodes where the file blocks will be copied eventually.
- The selection of IP addresses of DataNodes is purely randomized based on availability, replication factor and rack awareness that we have discussed earlier.
- Let's say the replication factor is set to default i.e. 3. Therefore, for each block the NameNode will be providing the client a list of (3) IP addresses of DataNodes. The list will be unique for each block.
- Suppose, the NameNode provided following lists of IP addresses to the client:
 - For Block A, list A = {IP of DataNode 1, IP of DataNode 4, IP of DataNode 6}
 - For Block B, set B = {IP of DataNode 3, IP of DataNode 7, IP of DataNode 9}
- Each block will be copied in three different DataNodes to maintain the replication factor consistent throughout the cluster.
- Now the whole data copy process will happen in three stages:



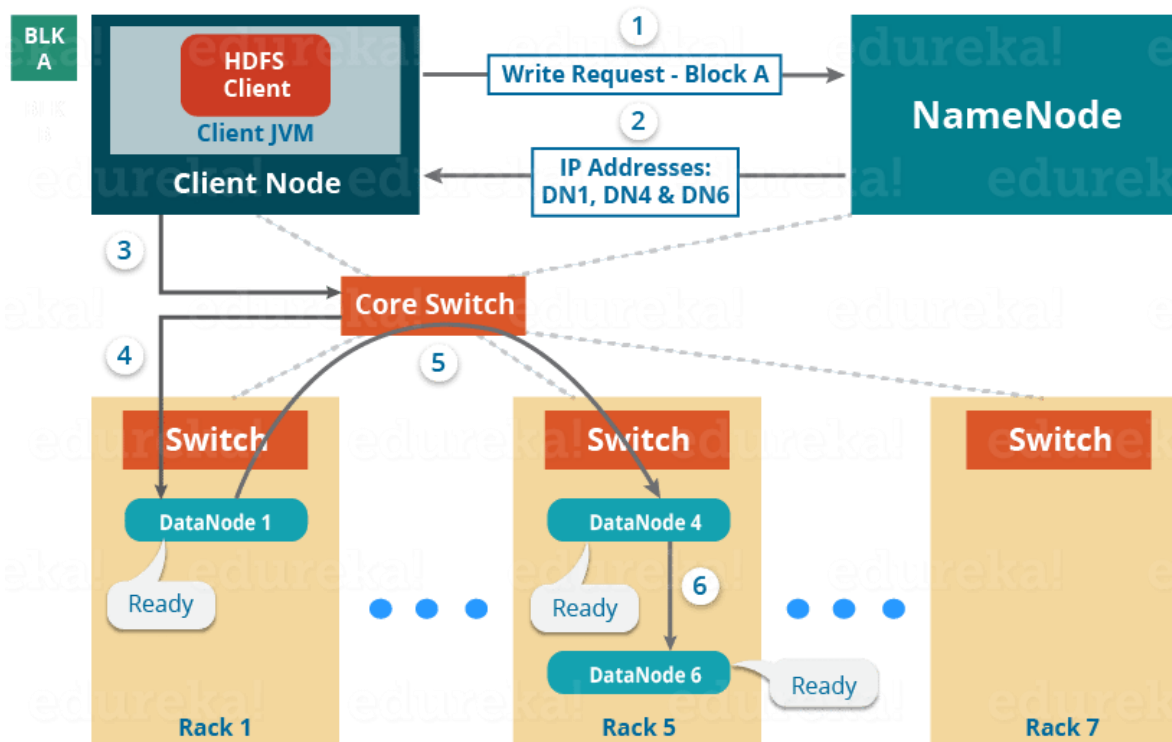
1. Set up of Pipeline
2. Data streaming and replication
3. Shutdown of Pipeline (Acknowledgement stage)

1. Set up of Pipeline:

Before writing the blocks, the client confirms whether the DataNodes, present in each of the list of IPs, are ready to receive the data or not. In doing so, the client creates a pipeline for each of the blocks by connecting the individual DataNodes in the respective list for that block. Let us consider Block A. The list of DataNodes provided by the NameNode is:

For Block A, list A = {IP of DataNode 1, IP of DataNode 4, IP of DataNode 6}.

Setting up HDFS - Write Pipeline

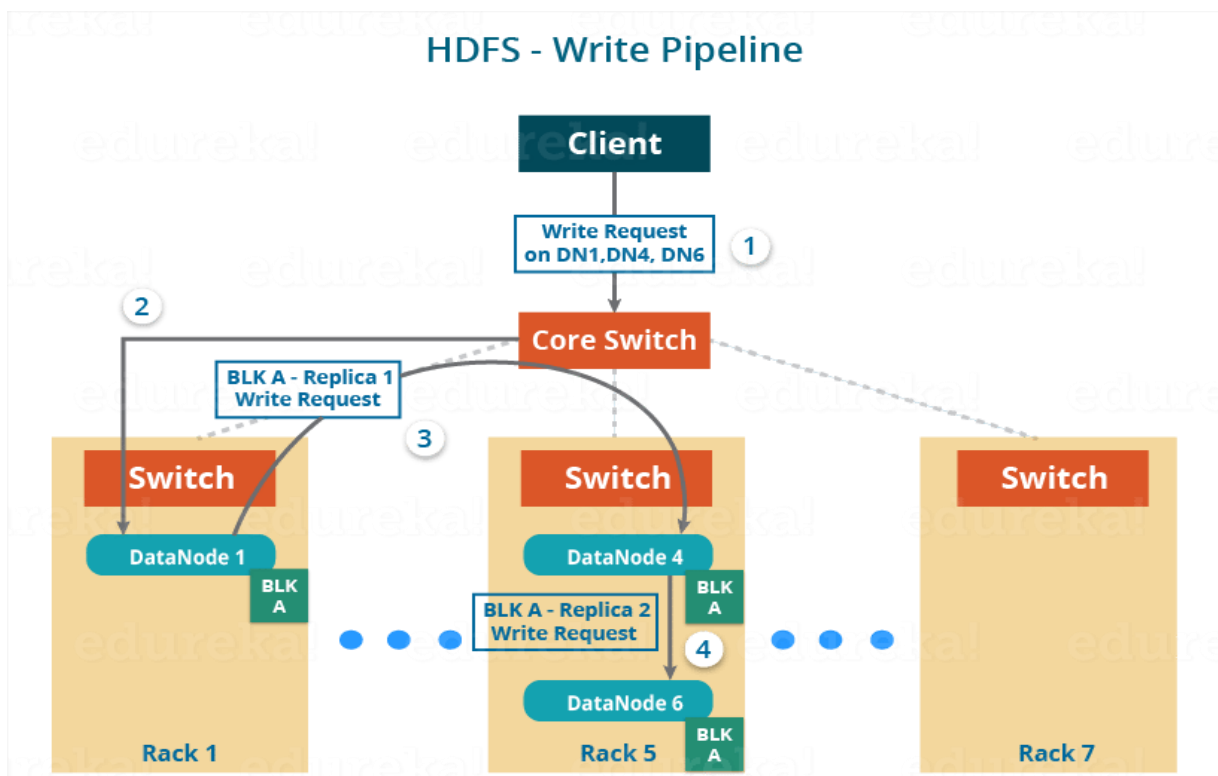


So, for block A, the client will be performing the following steps to create a pipeline:

- The client will choose the first DataNode in the list (DataNode IPs for Block A) which is DataNode 1 and will establish a TCP/IP connection.
- The client will inform DataNode 1 to be ready to receive the block. It will also provide the IPs of next two DataNodes (4 and 6) to the DataNode 1 where the block is supposed to be replicated.
- The DataNode 1 will connect to DataNode 4. The DataNode 1 will inform DataNode 4 to be ready to receive the block and will give it the IP of DataNode 6. Then, DataNode 4 will tell DataNode 6 to be ready for receiving the data.
- Next, the acknowledgement of readiness will follow the reverse sequence, i.e. From the DataNode 6 to 4 and then to 1.
- At last DataNode 1 will inform the client that all the DataNodes are ready and a pipeline will be formed between the client, DataNode 1, 4 and 6.
- Now pipeline set up is complete and the client will finally begin the data copy or streaming process.

2. Data Streaming:

As the pipeline has been created, the client will push the data into the pipeline. Now, don't forget that in HDFS, data is replicated based on replication factor. So, here Block A will be stored to three DataNodes as the assumed replication factor is 3. Moving ahead, the client will copy the block (A) to DataNode 1 only. The replication is always done by DataNodes sequentially.



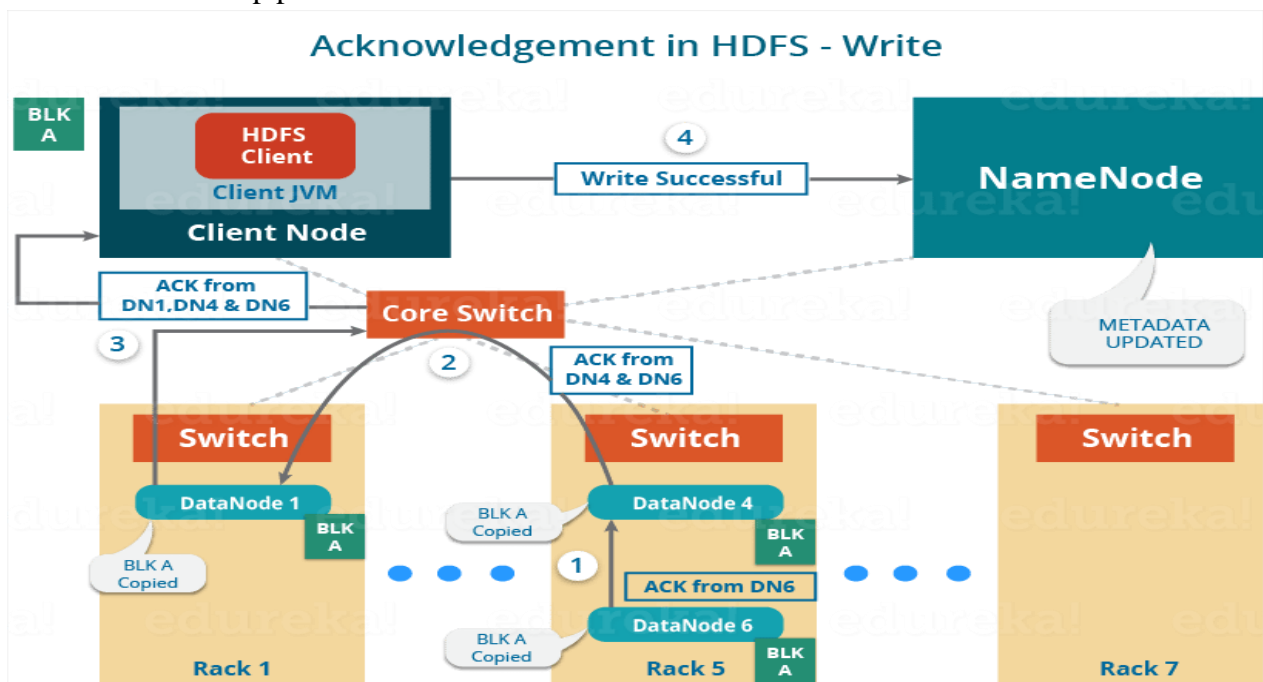
So, the following steps will take place during replication:

- Once the block has been written to DataNode 1 by the client, DataNode 1 will connect to DataNode 4.
- Then, DataNode 1 will push the block in the pipeline and data will be copied to DataNode 4.
- Again, DataNode 4 will connect to DataNode 6 and will copy the last replica of the block.

3. Shutdown of Pipeline or Acknowledgement stage:

Once the block has been copied into all the three DataNodes, a series of acknowledgements will take place to ensure the client and NameNode that the data has been written successfully. Then, the client will finally close the pipeline to end the TCP session.

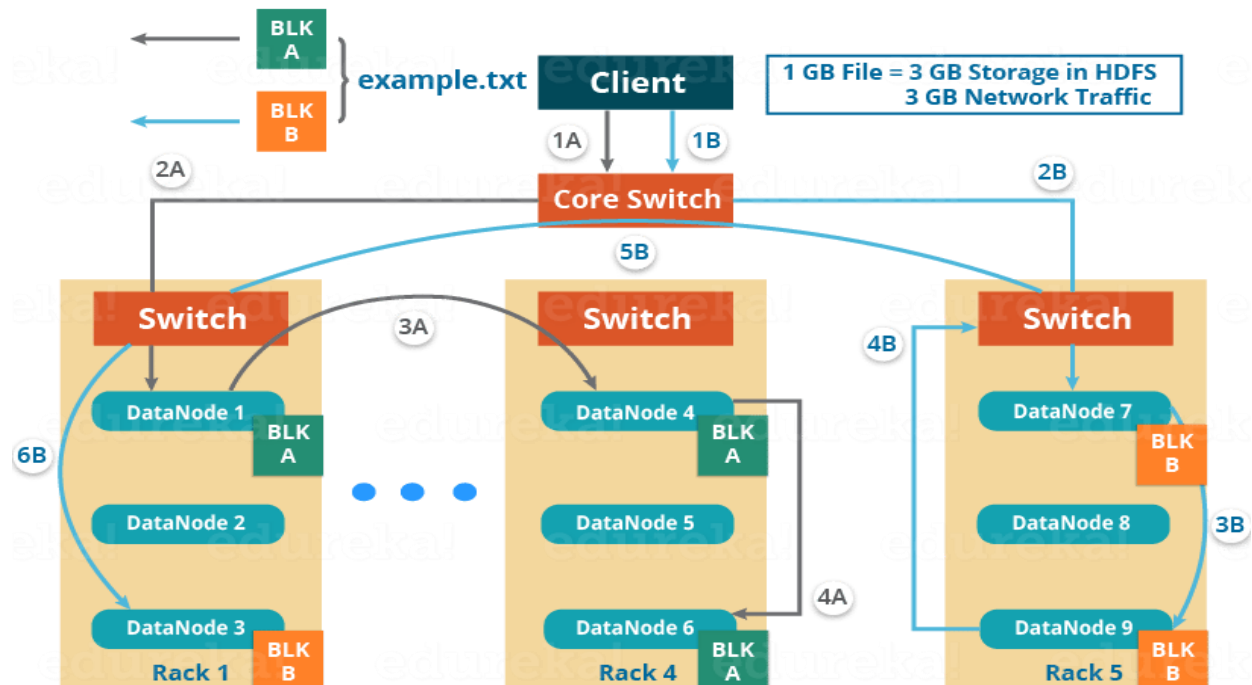
As shown in the figure below, the acknowledgement happens in the reverse sequence i.e. from DataNode 6 to 4 and then to 1. Finally, the DataNode 1 will push three acknowledgements (including its own) into the pipeline and send it to the client. The client will inform NameNode that data has been written successfully. The NameNode will update its metadata and the client will shut down the pipeline.



Similarly, Block B will also be copied into the DataNodes in parallel with Block A. So, the following things are to be noticed here:

- The client will copy Block A and Block B to the first DataNode **simultaneously**.
- Therefore, in our case, two pipelines will be formed for each of the block and all the process discussed above will happen in parallel in these two pipelines.
- The client writes the block into the first DataNode and then the DataNodes will be replicating the block sequentially.

HDFS Multi - Block Write Pipeline

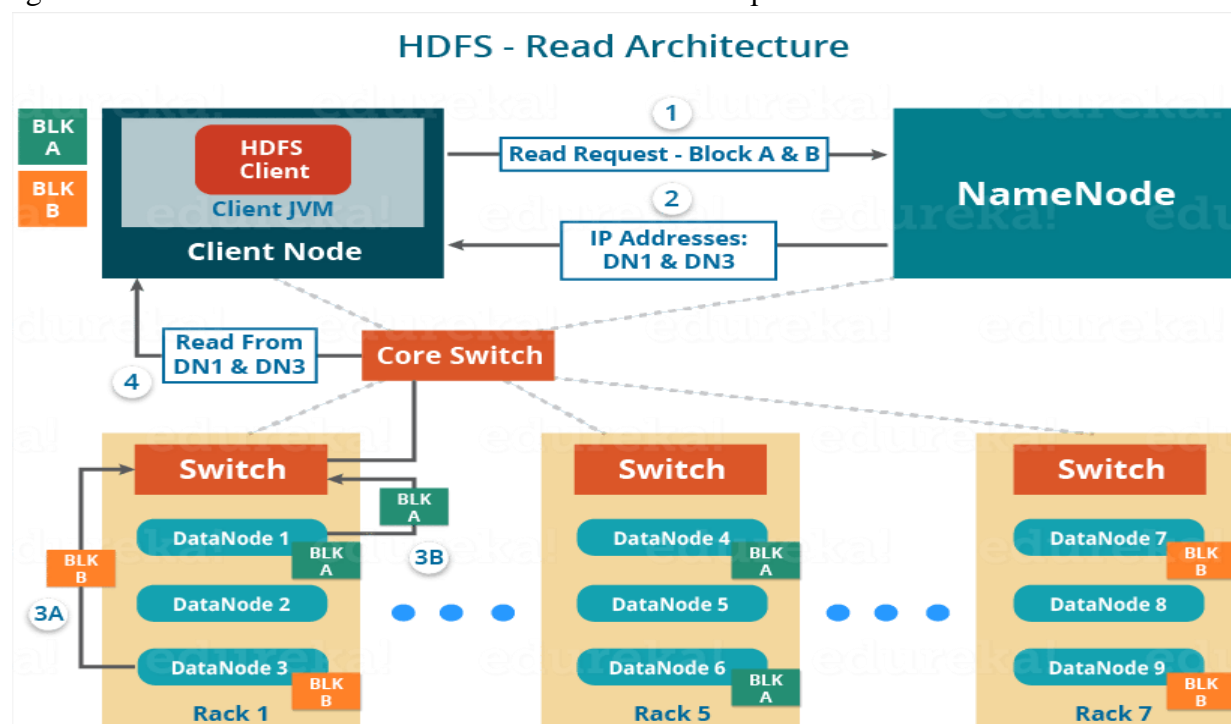


As you can see in the above image, there are two pipelines formed for each block (A and B). Following is the flow of operations that is taking place for each block in their respective pipelines:

- For Block A: 1A -> 2A -> 3A -> 4A
- For Block B: 1B -> 2B -> 3B -> 4B -> 5B -> 6B

HDFS Read Architecture:

HDFS Read architecture is comparatively easy to understand. Let's take the above example again where the HDFS client wants to read the file "example.txt" now.



Now, following steps will be taking place while reading the file:

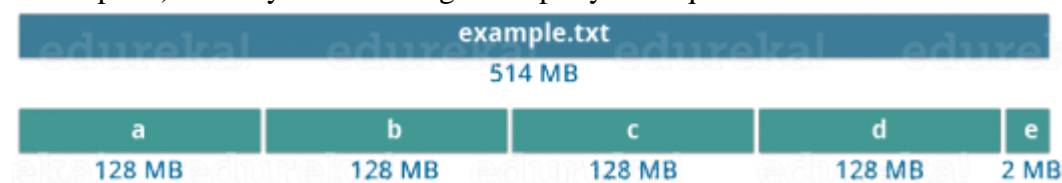
- The client will reach out to NameNode asking for the block metadata for the file “example.txt”.
- The NameNode will return the list of DataNodes where each block (Block A and B) are stored.
- After that client, will connect to the DataNodes where the blocks are stored.
- The client starts reading data parallel from the DataNodes (Block A from DataNode 1 and Block B from DataNode 3).
- Once the client gets all the required file blocks, it will combine these blocks to form a file.

While serving read request of the client, HDFS selects the replica which is closest to the client. This reduces the read latency and the bandwidth consumption. Therefore, that replica is selected which resides on the same rack as the reader node, if possible.

16. what is a block and how is it formed? And explain about block replacement using Rack Awareness algorithm.

Ans:

Blocks are the nothing but the smallest continuous location on your hard drive where data is stored. In general, in any of the File System, you store the data as a collection of blocks. Similarly, HDFS stores each file as blocks which are scattered throughout the Apache Hadoop cluster. The default size of each block is 128 MB in Apache Hadoop 2.x (64 MB in Apache Hadoop 1.x) which you can configure as per your requirement.



It is not necessary that in HDFS, each file is stored in exact multiple of the configured block size (128 MB, 256 MB etc.). Let’s take an example where I have a file “example.txt” of size 514 MB as shown in above figure. Suppose that we are using the default configuration of block size, which is 128 MB. Then, how many blocks will be created? 5, Right. The first four blocks will be of 128 MB. But, the last block will be of 2 MB size only.

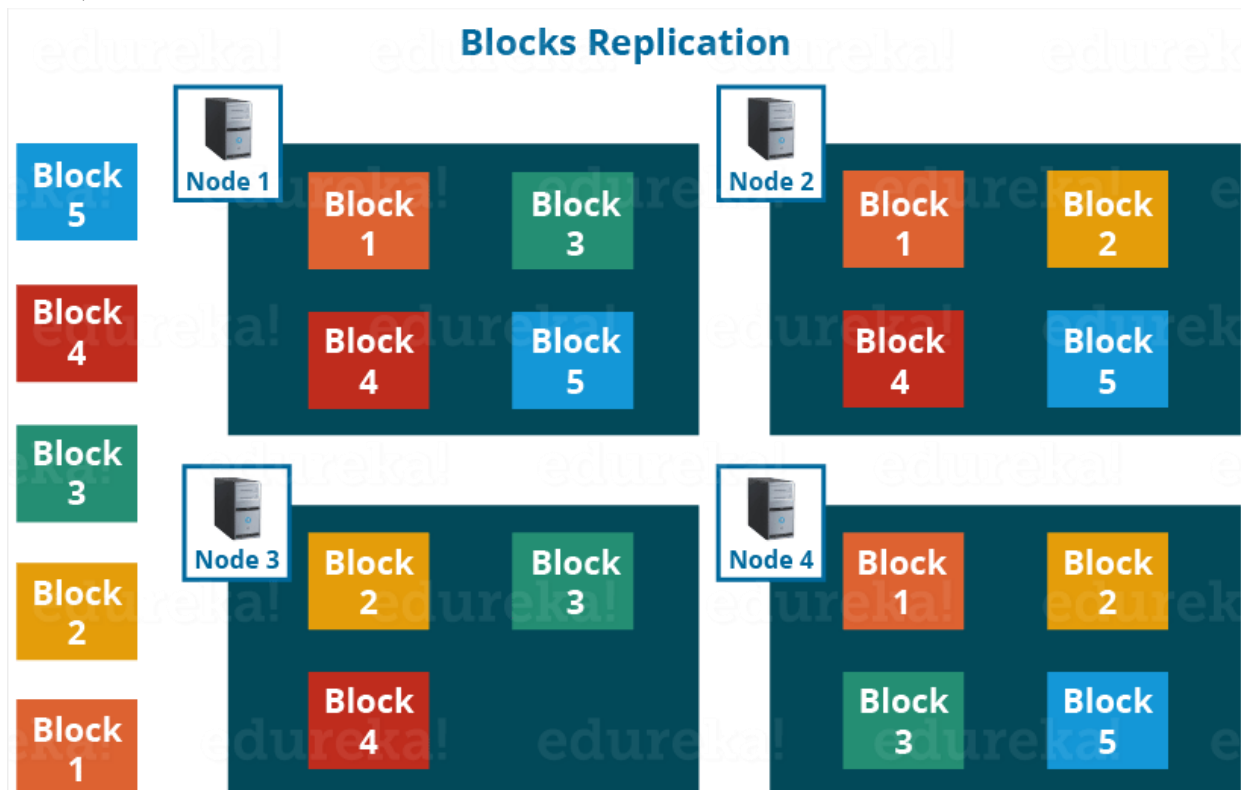
Now, you must be thinking why we need to have such a huge blocks size i.e. 128 MB?

Well, whenever we talk about HDFS, we talk about huge data sets, i.e. Terabytes and Petabytes of data. So, if we had a block size of let’s say of 4 KB, as in Linux file system, we would be having too many blocks and therefore too much of the metadata. So, managing these no. of blocks and metadata will create huge overhead, which is something, we don’t want.

Replication Management:

HDFS provides a reliable way to store huge data in a distributed environment as data blocks. The blocks are also replicated to provide fault tolerance. The default replication factor is 3 which is again configurable. So, as you can see in the figure below where each block is replicated three times and stored on different DataNodes (considering the default replication

factor):

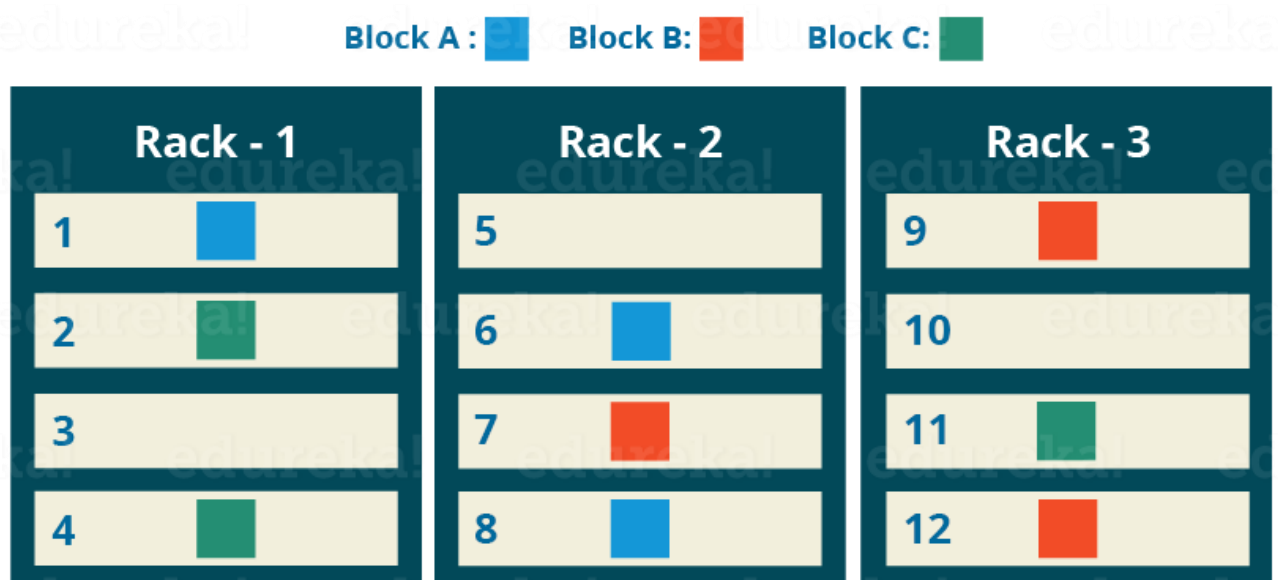


Therefore, if you are storing a file of 128 MB in HDFS using the default configuration, you will end up occupying a space of 384 MB (3×128 MB) as the blocks will be replicated three times and each replica will be residing on a different DataNode.

Note: The NameNode collects block report from DataNode periodically to maintain the replication factor. Therefore, whenever a block is over-replicated or under-replicated the NameNode deletes or add replicas as needed.

Rack Awareness:

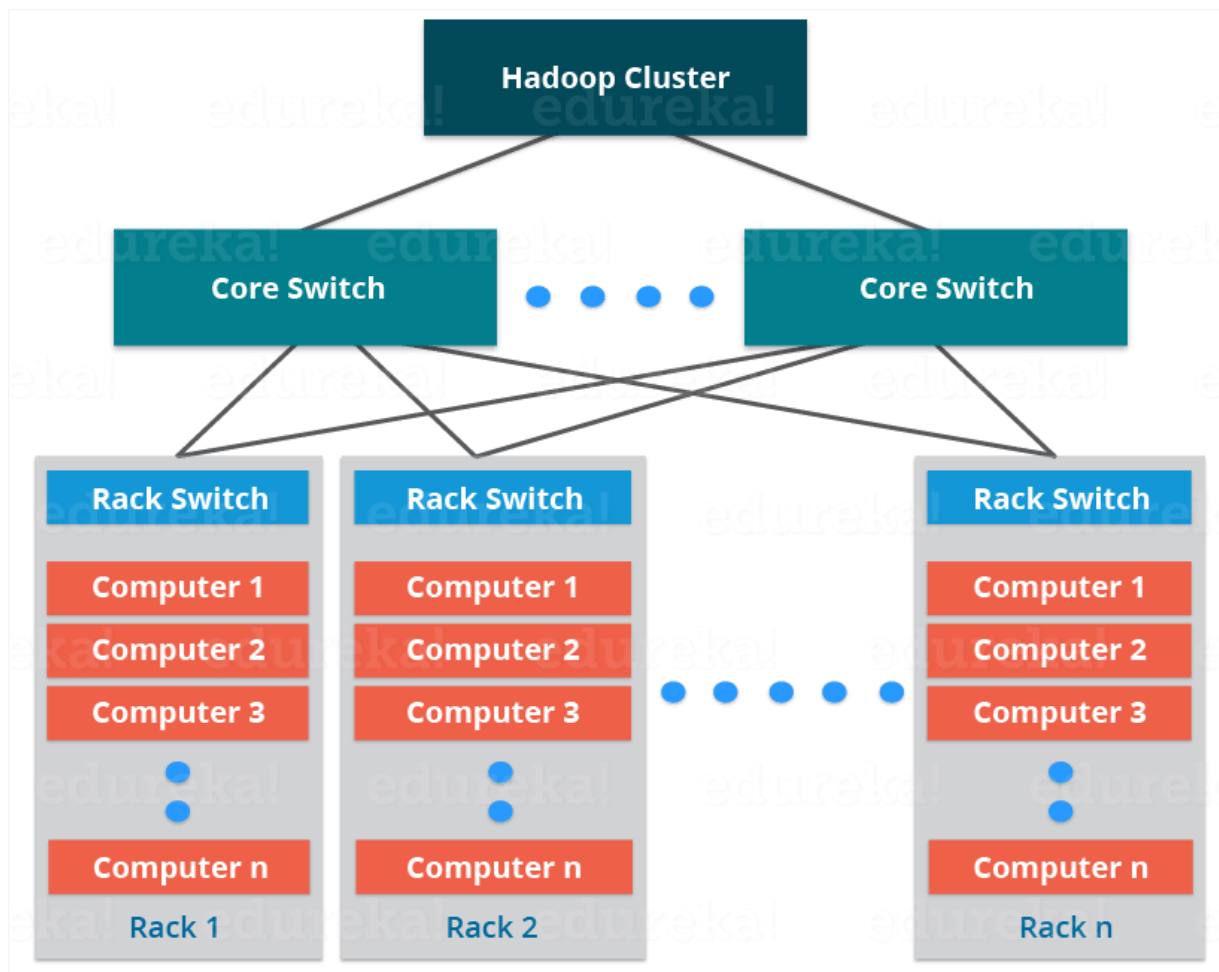
Rack Awareness Algorithm



Anyways, moving ahead, let's talk more about how HDFS places replica and what is rack awareness? Again, the NameNode also ensures that all the replicas are not stored on the same rack or a single rack. It follows an in-built Rack Awareness Algorithm to reduce latency as well as provide fault tolerance. Considering the replication factor is 3, the Rack Awareness Algorithm says that the first replica of a block will be stored on a local rack and the next two replicas will be stored on a different (remote) rack but, on a different DataNode within that

(remote) rack as shown in the figure above. If you have more replicas, the rest of the replicas will be placed on random DataNodes provided not more than two replicas reside on the same rack, if possible.

This is how an actual Hadoop production cluster looks like. Here, you have multiple racks populated with DataNodes:



Advantages of Rack Awareness:

So, now you will be thinking why do we need a Rack Awareness algorithm? The reasons are:

- **To improve the network performance:** The communication between nodes residing on different racks is directed via switch. In general, you will find *greater network bandwidth* between machines in the same rack than the machines residing in different rack. So, the Rack Awareness helps you to have reduce write traffic in between different racks and thus providing a better write performance. Also, you will be gaining increased read performance because you are using the bandwidth of multiple racks.
- **To prevent loss of data:** We don't have to worry about the data even if an entire rack fails because of the switch failure or power failure. And if you think about it, it will make sense, as it is said that *never put all your eggs in the same basket*.